Pergamon

Computers Math. Applic. Vol. 29, No. 6, pp. 47–57, 1995

0898-1221/95 $9.50 + 0.00

0898-1221(95)00006-2

Parallel Computation of Perfect Elimination Schemes Using Partition Techniques on Triangulated Graphs

S. D. NIKOLOPOULOS[1]
Department of Computer Science, University of Cyprus
75 Kallipoleos Str., CY-1678 Nicosia, Cyprus
stavros@turiug.cs.ucy.ac.cy

S. D. DANIELOPOULOS[2]
Division of Applied Mathematics and Informatics
University of Ioannina, GR-451 10 Ioannina, Greece

(Received April 1994; accepted May 1994)

Abstract—Perfect elimination schemes (p.e.s.) occur in a number of important problems such as perfect Gaussian elimination. The main objective of this paper is to study the parallel computation of p.e.s. of a triangulated or perfect elimination graph $G = (V, E)$, with $n = |V|$ vertices. We start with the notion of partitioning a triangulated graph into a set of (mutually disjoint) adjacency-level sets and we present a parallel algorithm, based mainly on the properties of the adjacency-level sets, which computes a p.e.s. in time $O(\log L \cdot \log H)$ using $L \cdot H \cdot n^2$ processors on a CRCW-PRAM. The computation of the adjacency-level sets of a triangulated graph can be done in time $O(\log L)$ with $L \cdot H \cdot n^2$ processors within the same type of computational model. Here, $L < n$ and $H < n$ are the length and the height of the graph, respectively.

Keywords—Parallel algorithm, Perfect elimination, Graph partition, Lexicographic search, Triangulated graph, CRCW-PRAM, Complexity.

1. INTRODUCTION

Perfect elimination schemes (p.e.s.) occur in a number of important problems among which perfect Gaussian elimination. The manipulation of certain matrices, e.g., sparse symmetric matrices can be reduced to the manipulation of corresponding graphs. A graph possesses a p.e.s. if and only if it is *triangulated* (triangulated graphs have also been called *chordal, monotone transitive* and *perfect elimination graphs*).

Our objective is to study the parallel computation of a p.e.s. of a triangulated graphs $G = (V, E)$, with $n = |V|$ vertices. Naor, Naor and Schaffer [1] have given a number of parallel algorithms which, after computing first a number of other entities, eventually compute a p.e.s. in time $O(\log^3 n)$ with $O(n^4)$ processors. Ho and Lee [2] compute first a clique tree and, given the clique tree, a p.e.s. in overall time $O(\log n)$ with $O(n^4)$ processors.

The authors of this paper start with the notion of partitioning a graph into a set of (mutually disjoint) adjacency-level sets and compute a p.e.s., directly, in time $O(\log L \cdot \log H)$ with $O(L \cdot H \cdot n^2)$ processors. For the process of partitioning, we use a parallel algorithm which com-

[1]Author to whom all correspondence should be addressed.
[2]Professor Stylianos D. Danielopoulos passed away in 1992.

putes the adjacency-level sets of a graph in time $O(\log L)$ with $L \cdot H \cdot n^2$ processors [3], where $L < n$ and $H < n$ are the *length* and the *height* of the graph, respectively (see Section 3). The computational model used in all cases is a Concurrent-Read Concurrent-Write Parallel RAM (CRCW-PRAM).

Next, we establish the notation and terminology used here. We call the graph a pair $G = (V, E)$, where V is a finite set of $n = |V|$ elements called *vertices* and $E = \{(x, y) \mid x, y \in V, x \neq y\}$ a set of $e = |E|$ unordered vertex pairs called *edges*. The vertices that belong to an edge are said to be *adjacent*. We call the adjacency set of a vertex $x \in V$, which we denote by $\text{adj}(x)$, the set of all vertices that are adjacent to x, i.e., $\text{adj}(x) = \{y \mid y \in V \& (x, y) \in E\}$. We can extend the notion of adjacency set so that for any set $S \subseteq V$, we define $\text{adj}(S) = \{y \mid y \in \text{adj}(x) \ \& \ x \in S\} - S$. Given a subset $W \subseteq V$ of the vertices, we define the subgraph *induced* by W to be $G[W] = (W, E(W))$, where $E(W) = \{(x, y) \mid (x, y) \in E \ \& \ x \in W\}$. A graph is *complete* if every pair of distinct vertices is an edge. A subset W of the vertex set V, that induces a complete subgraph, is called a *clique*. A vertex $x \in V$ is called *simplicial* if its adjacency set $\text{adj}(x)$ is a clique [4].

A *chain* of *length* r in a graph G, is a sequence of distinct vertices $[v_0, v_1, \ldots, v_r]$, where $(v_{i-1}, v_i) \in E$, for $1 \leq i \leq r$. A cycle is a chain of length $r \geq 2$ such that the vertices v_0 and v_r are adjacent. A graph G is called *triangulated* (or *chordal*) if every cycle of length, at least, four has a chord, i.e., an edge joining two nonconsecutive vertices in a cycle [1,4–7].

An *ordering* of the vertices of set V of a graph $G = (V, E)$ is a bijection $\sigma : V \leftrightarrow \{1, 2, \ldots, n\}$, where $n = |V|$. An ordering $\sigma = [v_1, v_2, \ldots, v_n]$ is called a *perfect elimination scheme* (p.e.s.) if each v_i is a simplicial vertex of the induced subgraph $G[\{v_i, v_{i+1}, \ldots, v_n\}]$. A graph $G = (V, E)$ has a p.e.s. if and only if it is triangulated [4,8–10], (see Figure 1).

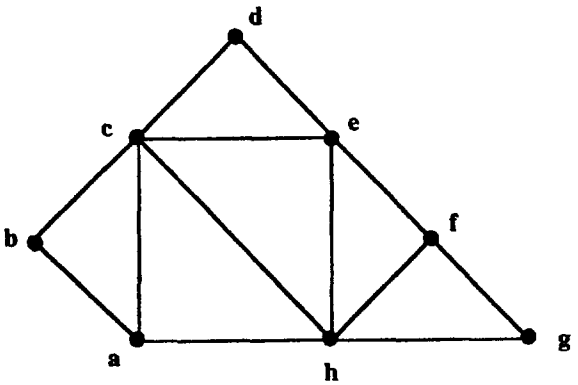


Figure 1. A triangulated graph; one p.e.s. is $\sigma = [g, f, d, e, h, a, b, c]$.

If $\sigma = [v_1, v_2, \ldots, v_n]$ is an ordering of the vertices of graph G, then we define the reverse ordering $\sigma_R = [v_n, v_{n-1}, \ldots, v_1]$. If we denote by σ_R an ordering $[v_1, v_2, \ldots, v_n]$ of the vertices of graph G, then $\sigma_R^{-1}(x)$ is the number (index) that shows the position of vertex x in the ordering σ_R. It is obvious that $1 \leq \sigma_R^{-1}(x) \leq n$.

2. SEQUENTIAL COMPUTATION OF A P.E.S.

Rose, Tarjan and Lueker [11], and Tarjan and Yannakakis [12,13] have presented algorithms for the sequential computation of a p.e.s. We shall focus our attention on the algorithm developed by Rose *et al.* [11], which is relevant to our work, and uses a lexicographic breadth-first search to compute an ordering σ of the vertices of the graph. The algorithm, which we call LEXBFS and is listed in Figure 2, numbers the nodes of the graph, from n to 1, in the order in which they are selected in step "Select" of the algorithm. This numbering fixes the positions of the vertices in the ordering σ. The label of each vertex v consists of a string of digits which are concatenated in decreasing order. The vertices are lexicographically ordered according to their labels. In cases in which more than one of the vertices have the same label, the selection of the vertex can be

made arbitrarily. The algorithm LEXBFS computes a p.e.s. of a triangulated graph $G = (V, E)$ in time $O(n + e)$, where $n = |V|$ and $e = |E|$.

Algorithm LEXBFS
begin
1. Select an arbitrary vertex v from the graph $G = (V, E)$, $v \in V$; p.e.s.
   Assign the label "n" to vertex v, where $n = |V|$, and the empty label λ to each vertex $x \in V - \{v\}$, i.e., Label$(v) \leftarrow$ "n", Label$(x) \leftarrow \lambda$, $\forall x \in V - \{v\}$;
2. **for** $i \leftarrow n$ **to** 1 **step** -1 **do**
   Select : pick an unnumbered vertex v with largest label;
   Labelling : Label $(v) \leftarrow$ Label$(v) \parallel$ "i";
   Numbering : $\sigma[i] \leftarrow v$; // this assigns to vertex v the number i //
   Update : **for** each unnumbered vertex $w \in$ adj(v) **do**
   Label$(w) \leftarrow$ Label$(w) \parallel$ "$i - 1$";
   end;
end.

Figure 2. The Algorithm LEXBFS.

We notice that the process of numbering a specific vertex involves only the adjacency set of that vertex. This leads us to consider a partition of the set V of vertices into a collection of subsets which we call *adjacency-level sets* or simply *adjacency levels.* Such a partition is defined with respect to a specific vertex. There are therefore, $n = |V|$ possible partitions of a graph.

3. ADJACENCY LEVEL SETS

Given a graph $G = (V, E)$ and a vertex $v \in V$, we define a partition $\Delta(G, v)$ of the vertex set V (we shall frequently use the term *partition of the graph* G), with respect to the vertex v in the following manner:

$$\Delta(G, v) = \{AL(v, \ell) \mid v \in V,\ 0 \leq \ell \leq L_v,\ 1 \leq L_v < |V|\},$$

where $AL(v, \ell)$, $0 \leq \ell \leq L_v$, are the adjacency-level sets, or simply the adjacency levels, and L_v is the length of the partition $\Delta(G, v)$. The adjacency-level sets have the following properties:

$$\begin{aligned} AL(v, \ell) \cap AL(v, \ell') &= \emptyset, && \text{for } \ell \neq \ell', \\ \text{adj}(x) \cap AL(v, \ell - 1) &\neq \emptyset, && \forall x \in AL(v, \ell), \quad 1 \leq \ell \leq L_v, \\ \text{adj}(x) \cap AL(v, \ell - 2) &= \emptyset, && \forall x \in AL(v, \ell), \quad 2 \leq \ell \leq L_v \end{aligned}$$

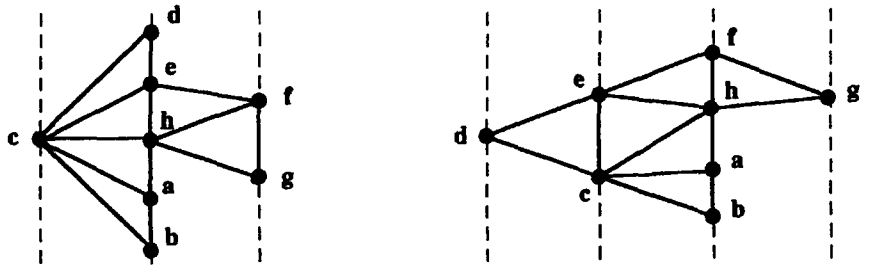
and

$$\bigcup_{0 \leq \ell \leq L_v} AL(v, \ell) = V.$$

The adjacency-level sets of the partition $\Delta(G, v)$, are defined recursively as follows:

$$\begin{aligned} AL(v, 0) &= \{v\}, \qquad v \in V, \\ AL(v, 1) &= \text{adj}(v), \qquad \text{and} \\ AL(v, l) &= \{y \mid y \in \text{adj}(AL(v, \ell - 1)) - AL(v, \ell - 1) - AL(v, \ell - 2)\}, \qquad \ell \leq 2. \end{aligned}$$

Thus, the adjacency-level sets of the graph of Figure 1, with respect to vertices c and g, are shown in Figure 3(a) and Figure 3(b), respectively. The lengths L_c and L_d are 2 and 3, respectively.



(a) (b)

Figure 3. Partitions of the graph of Figure 1, with respect to vertices c ($L = 2$) and d ($L = 3$), respectively.

To a given graph $G = (V, E)$, we associate a length L defined by

$$L = \max\{L_i \mid L_i \text{ is the length of } \Delta(G, x_i),\ x_i \in V\}, \qquad \text{for } |V| > 1,$$
$$L = 1, \qquad \text{for } |V| = 1,$$

and a height H defined by

$$H = \max\{x \mid x = |AL(x, \ell)|,\ \ell = 0, 1, \ldots, L\}.$$

We shall call the magnitudes L and H defined above, *the length and the height of graph* $G = (V, E)$, respectively. Clearly, $L_x \leq L$, for every $x \in V$.

We point out here that the adjacency-level sets $AL(v, \ell), 0 \leq \ell \leq L_v$, of partition (G, v), can also be computed by considering first the distance matrix of the graph G and then extracting all necessary set information from it [14].

4. PARALLEL COMPUTATION OF A P.E.S.

We start with some observations on algorithm LEXBFS, discussed in Section 2, which computes a p.e.s. of a triangulated graph $G = (V, E)$ in a sequential environment. The algorithm involves two main processes, strongly related or dependent on each other, which are the *process of labelling* and the *process of numbering.* As mentioned in the appropriate section, numbering fixes the positions of the vertices in the ordering σ. We denote the label of a vertex $x \in V$ by Label(x), and the number assigned to it by a_x, where a_x is an integer in the range 1 to $n = |V|$.

We observe that, if a_x, a_y are the numbers assigned to vertices x, $y \in V$, respectively, and $a_x > a_y$, then Label(x) > Label(y). This holds for every pair of vertices x, $y \in V$. We observe, moreover, that the number a_x assigned to vertex x is equal to the number of vertices which have label smaller than Label(x), plus 1. Therefore, the processes of labelling and numbering in algorithm LEXBFS can be executed independently, i.e., first all vertices are labelled and then are numbered. This is true because, if we label first all vertices of the graph using the process of labelling, then we can number the vertices using the statement $\sigma[a_x] \leftarrow x$ (this assigns number a_x to vertex x), where a_x = the number of vertices with a label smaller than Label(x) plus 1. This is one of the facts on which is based the parallel computation of a p.e.s. It is obvious that, even the processes of labelling and numbering are executed separately, the complexity of algorithm LEXBFS remains unchanged, $O(|V| + |E|)$.

We apply now algorithm LEXBFS to a partitioned graph $\Delta(G, v)$, $v \in V$, with adjacency-level sets $AL(v, 0)$, $AL(v, 1), \ldots, AL(v, \ell), \ldots, AL(v, L)$. We start the numbering with the vertex of level zero ($\ell = 0$), i.e., Label(v) $\leftarrow$ "$n + 1$" and Label(x) $\leftarrow \lambda$ $\forall x \in V - \{v\}$, initially.

According to the process of labelling and numbering of the algorithm LEXBFS (steps "Select" through "Update"), the numbering of the vertices of Level ℓ cannot start, unless all vertices of Level $\ell-1$, $1 \leq \ell \leq L$, are numbered. Therefore, Label(x) > Label(y) for each pair of vertices x and y, where $x \in AL(v,\ell)$, $y \in AL(v,\ell')$ and $\ell < \ell'$.

Next, we present a lemma, which describes a property f triangulated graphs.

LEMMA 4.1. *Consider a triangulated graph $G = (V,E)$ and its adjacency-level sets $AL(v,0)$, $AL(v,1),\ldots,AL(v,L)$, $v \in V$. Let x,y,z be vertices of $AL(v,\ell)$, $1 \leq \ell \leq L$, such that $(x,y) \in E$, $(y,z) \in E$, $(x,z)E$, and*

$$|\text{adj}(x) \cap AL(v,\ell-1)| \geq \max\{|\text{adj}(y) \cap AL(v,\ell-1)|,\ |\text{adj}(z) \cap AL(v,\ell-1)|\}.$$

Then the following relation holds:

$$\text{adj}(x) \cap AL(v,\ell-1) \supseteq \text{adj}(y) \cap AL(v,\ell-1) \supseteq \text{adj}(z) \cap AL(v,\ell-1)$$

and, therefore,

$$|\text{adj}(x) \cap AL(v,\ell-1)| \geq |\text{adj}(y) \cap AL(v,\ell-1)| \geq |\text{adj}(z) \cap AL(v,\ell-1)|.$$

PROOF. Consider the vertex u such that $u \in \text{adj}(y) \cap AL(v,\ell-1)$ and $u \notin \text{adj}(x) \cap AL(v,\ell-1)$. Then $(u,x) \notin E$. Since $|\text{adj}(x) \cap AL(v,\ell-1)| \geq |\text{adj}(y) \cap AL(v,\ell-1)|$, there follows that there is a vertex $w \in \text{adj}(x) \cap AL(v,\ell-1)$ such that $(w,y) \notin E$. Therefore, we are led to the conclusion that graph $G[AL(v,0) \cup \cdots \cup AL(v,\ell-1) \cup AL(v,\ell)]$ contains a cycle of length greater than 3, which is absurd. Therefore, $\text{adj}(x) \cap AL(v,\ell-1) \supseteq \text{adj}(y) \cap AL(v,\ell-1)$. In a similar manner, it is shown that $\text{adj}(x) \cap AL(v,\ell-1) \supseteq \text{adj}(z) \cap AL(v,\ell-1)$.

Assume now that $u \in \text{adj}(z) \cap AL(v,\ell-1)$ and $u \notin \text{adj}(y) \cap AL(v,\ell-1)$. Since $\text{adj}(x) \cap AL(v,\ell-1) \supseteq \text{adj}(z) \cap AL(v,\ell-1)$, there follows that $u \in \text{adj}(x) \cap AL(v,\ell-1)$. This implies that the cycle $[u,x,y,z]$ has a length greater than 3, which is absurd, hence, $\text{adj}(x) \cap AL(v,\ell-1) \supseteq \text{adj}(y) \cap AL(v,\ell-1) \supseteq \text{adj}(z) \cap AL(v,\ell-1)$. Therefore,

$$|\text{adj}(x) \cap AL(v,\ell-1)| \geq |\text{adj}(y) \cap AL(v,\ell-1)| \geq |\text{adj}(z) \cap AL(v,\ell-1)|,$$

for $\ell = 1,2,\ldots,L$. ∎

The above facts lead us to the following observations about algorithm LEXBFS, when it is applied to a partitioned graph $\Delta(G,v)$, $v \in V$.

OBSERVATION 1. Label(x) > Label(y) > Label(z), $\forall x \in AL(v,\ell)$, $\forall y \in AL(v,\ell)$ and $\forall y \in AL(v,\ell)$, where $\ell < \ell' < \ell''$.

OBSERVATION 2. Vertex $x \in AL(v,\ell)$ is numbered before $y \in AL(v,\ell')$ is numbered, if $\ell < \ell'$.

OBSERVATION 3. The vertex that is selected and numbered first in set $AL(v,\ell)$, is that vertex which has the largest label among unnumbered vertices, i.e., the vertex which has the largest number of adjacent vertices in set $AL(v,\ell-1)$, $1 \leq \ell \leq L$, (Lemma 4.1).

Let $AL(v,0), AL(v,1),\ldots,AL(v,L)$ be the adjacency-level sets of a partitioned graph $\Delta(G,v)$, $v \in V$. Based on the previous facts and observations, mainly on the independence of labelling and numbering processes, we can select and label, in parallel, all vertices of graph in such a way that Label(x) > Label(y), for every $x \in AL(v,\ell)$ and for every $y \in AL(v,\ell')$, with $\ell < \ell'$. It is easy to establish this property, if we label all vertices of level ℓ with the same label, say i, and all vertices of the next higher level $\ell+1$ with the same label, say j, smaller than i, $0 \leq \ell < L$. It is clear that, in the process of numbering, vertex $x \in AL(v,\ell)$ is numbered before $y \in AL(v,\ell')$ is numbered, $\ell < \ell'$, i.e., $\sigma^{-1}(x) > \sigma^{-1}(y)$. Therefore, an ordering $\sigma = [AL(v,L), AL(v,L-1),\ldots,AL(v,0)]$ is converted into a p.e.s. if the vertices of each set $AL(v,\ell), l = 1,2,\ldots,L$ are appropriately reordered.

According to Observation 3, the vertex which selected and numbered first in the set $AL(v,\ell)$, is always that which has the largest label among unnumbered vertices, i.e., the vertex which has the largest number of adjacent vertices in set $AL(v,\ell-1)$, $1 \leq \ell \leq L$. Let x be the vertex which is numbered first in the set $AL(v,\ell)$, $1 \leq \ell \leq L$. It is obvious that by applying algorithm LEXBFS, the vertices which are numbered next are the vertices of set $AL(v,\ell)$, which are adjacent to vertex x, i.e., the vertices of set $\text{adj}(x) \cap AL(v,\ell)$. This is true because the vertices that belong to this set have the next largest labels of any other vertex of set $AL(v,\ell)$, since they are adjacent to more vertices of set $AL(v,\ell-1)$ (Lemma 4.1).

The above observations lead us to partition graph $G[AL(v,\ell)]$ with respect to vertex $x \in AL(v,\ell)$, $1 \leq \ell \leq L$, which is adjacent to the largest number of vertices that have a larger label than x, and to decrease the labels of partition $\Delta(AL(v,\ell),x)$ so that $\text{Label}(y) > \text{Label}(z)$, for every $y \in AL(x,\ell)$ and for every $z \in AL(x,\ell')$, with $\ell < \ell'$.

An important point in the process of labelling of a partitioned graph $\Delta(G,v), v \in V$, is that the labels of vertices decrease (lexicographically) as the value of ℓ increases, i.e., $\text{Label}(x_0) > \text{Label}(x_1) > \cdots > \text{Label}(x_L)$, for every $x_i \in AL(v,i), 0 \leq i \leq L$. A natural way, and more convenient to us, to label the vertices of the partitioned graph is to assign the value of level ℓ to the label of vertices in level ℓ, i.e., assign the value ℓ to the label of vertices in set $AL(x,\ell)$, $0 \leq \ell \leq L$. By this, we establish the property $\text{Label}(x_0) < \text{Label}(x_1) < \cdots < \text{Label}(x_L)$, for every $x_i \in AL(v,i)$, $0 \leq i \leq L$. Now, the vertex which is selected first in set $AL(v,\ell)$, say x, is that which is adjacent to the largest number of vertices with a smaller label than x, $1 \leq \ell \leq L$. This does not cause any serious modification in the facts and observations listed above. The only deference is that the process of numbering produces, now, the reverse ordering σ_R instead of σ. The reverse ordering of σ_R produces the ordering σ.

Next, we define two sets of vertices for each vertex $x \in V$ of graph $G(V,E)$, which we call $\text{Min}(x)$ and $\text{Equal}(x)$.

Min(x) is defined as the set which contains all the vertices that have a label which is smaller than the label of vertex $x \in V$, i.e.,

$$\text{Min}(x) = \{y \mid \text{Label}(y) < \text{Label}(x), \text{ for each } y \in V\}.$$

Equal(x) is defined to be the set that contains all the vertices of the graph which have a label equal to the label of vertex $v \in V$, i.e.,

$$\text{Equal}(x) = \{y \mid \text{Label}(y) = \text{Label}(x), \text{ for each } y \in V\}.$$

In addition to the above, we define a set, named SV, which contains vertices with respect to which various subgraphs are partitioned.

Based on the previous facts and observations, we proceed now to formulate an algorithm for the parallel computation of a perfect elimination scheme of a triangulated graph.

(1) First, the empty label is assigned to every vertex of graph $G(V,E)$, vertex v is selected, and sets SV and $\text{Equal}(v)$ are assigned initial values, i.e., $SV \leftarrow \{v\}$; $\text{Equal}(v) \leftarrow V$.

(2) Graph $G[\text{Equal}(v)]$ is partitioned, with respect to start vertex $v \in SV$, and the adjacency-level sets $AL(v,0), AL(v,1), \ldots, AL(v,L_v)$ are constructed. Then the label of each vertex $x \in AL(v,\ell)$ is updated according to the assignment $\text{Label}(x) \leftarrow \text{Label}(x) \| \text{“}\ell\text{”}$, $1 \leq \ell \leq L_v$.

(3) The vertex sets $\text{Min}(x)$ and $\text{Equal}(x)$ are computed, for every vertex $x \in V$. Moreover, for each vertex $x \in V$ the set C_x is computed such that it contains all the vertices that join vertex $x \in V$ by an edge and have a label smaller than that of vertex x_V, i.e., $C_x = \text{adj}(x) \cap \text{Min}(x)$.

(4) Next, the vertex x of set $AL(v,\ell)$, $1 \leq \ell \leq L_v$, which is adjacent to the largest number of vertices that have smaller labels than x, is determined. This is the vertex that has an adjacency set of largest cardinality contained in $\text{Min}(x)$.

(5) If $\text{Label}(x) = \text{Label}(y)$ for some vertices x, $y \in V$, or equivalently, if $|AL(v,\ell)| > 1$, for some ℓ, $1 \leq \ell \leq L_v$, then, Steps 2 through 4 are executed for the graph $G[AL(v,\ell)] = G[\text{Equal}(x)]$, with start vertex x (determined in Step 4).

(6) In Steps 2, 3, and 4, the labels of the vertices of the graph which are such that $\text{Label}(x) \neq \text{Label}(y)$, $\forall x, y \in V$, are computed. In this step, vertex $x \in V$ is numbered with the integer αx, which represents the number of vertices with a label smaller than the label of x, plus one, i.e., $\alpha_x = |\text{Min}(x)| + 1$, $\forall x \in V$.

In Figure 4, we give a more formal listing of Algorithm PARPES (PARallel computation of a Perfect Elimination Scheme) which is based on the previous method.

Algorithm PARPES
begin
1. **for** each $x \in V$ **do in parallel**
   label$(x) \leftarrow \lambda$; // assign empty label //
   end;
   $SV \leftarrow \{v\}$; Equal$(v) \leftarrow V$;
2. **for** each $v \in SV$ **do in parallel**
2.1 Partition the graph $G[\text{Equal}(v)]$, with respect to vertex $v \in SV$;
   // Let $AL(v,0), AL(v,1), \ldots, AL(v,L_v)$ be the a.l.s //
2.2 **for** each $x \in AL(v,\ell)$, $0 \leq \ell \leq L_v$, **do in parallel**
   Label$(x) \leftarrow$ Label$(x) \parallel$ "ℓ";
   end;
   end;
3. **for** each $x \in V$, **do in parallel**
3.1 **for** each $y \in V - \{x\}$, **do in parallel**
   if Label$(y) <$ Label(x) **then** Min$(x) \leftarrow \{y\}$;
   if Label$(y) =$ Label(x) **then** Equal$(x) \leftarrow \{y\}$;
   end;
3.2 $C_x \leftarrow \text{adj}(x) \cap \text{Min}(x)$;
   end;
4. **for** each $v \in SV$, **do in parallel**
   for each set $AL(v,\ell)$, $1 \leq \ell \leq L_v$, **do in parallel**
   Select a vertex $x \in AL(v,\ell)$ such that:
   $C_y \subseteq C_x$ or $C_y \cap C_x = \emptyset$, $\forall y \in AL(v,\ell)$;
   $SV \leftarrow \{x\}$; // set SV receivers one vertex from each level ℓ //
   end;
   end;
5. **if** Label$(x) =$ Label(y) for some vertices $x, y \in V$ **then** go to Step 2;
6. **for** each $x \in V$, **do in parallel**
   $a_x \leftarrow |\text{Min}(x)| + 1$;
   $\sigma_R[a_x] \leftarrow x$;
   $\sigma[x] \leftarrow \sigma_R[n + 1 - x]$;
   end;
end.

Figure 4. The Algorithm PARPES.

The execution of Step 2 of the algorithm is repeated until all the vertices of the graph are assigned different labels, i.e., $\text{Label}(x) \neq \text{Label}(y)$ for each pair of vertices x, $y \in V$.

THEOREM 4.1. *Given a triangulated graph $G = (V, E)$, the ordering σ computed by algorithm PARPES is a perfect elimination scheme (p.e.s.).*

PROOF. The correctness of Algorithm LEXBFS, Lemma 4.1, and the manner in which the label of each vertex is computed, prove that the ordering σ is a p.e.s. ∎

5. THE COMPLEXITY OF THE ALGORITHM

The computational model used in this paper is the well-known Concurrent-Read, Concurrent-Write Parallel RAM model (CRCW-PRAM) [15–18].

We obtain the complexity of the algorithm by computing the complexity of each step separately.

STEP 1. The assignment of values to n variables is performed in constant time $O(1)$ with n processors.

STEP 2. This step consists of two substeps which are executed k times. The value of k is determined subsequently.

SUBSTEP 2.1. The partition of graph $G[\text{Equal}(v)]$, $\text{Equal}(v) \subseteq V$, is executed in time $O(\log L)$ with $L \cdot H \cdot n^2$ processors (Algorithm ALS_2 of [3]). This holds during the first partition when $SV = \{v\}$ and $\text{Equal}(v) = V$. It is obvious that subsequent partitions of subgraphs induced by subsets of V, take less time than $O(\log L)$. It will be shown that the number of processors required during the repetition of this substep is smaller than the initial number of processors $L \cdot H \cdot n^2$ (see Theorem 5.1).

SUBSTEP 2.2. In this substep, the labels of n vertices are computed. This takes constant time $O(1)$ and n processors.

STEP 3. This step consists of Substeps 3.1 and 3.2.

SUBSTEP 3.1. Here the vertex sets $\text{Min}(x)$ and $\text{Equal}(x)$ are computed for each $x \in V$. This takes time $O(1)$ with n^2 processors.

SUBSTEP 3.2. In this substep, n set intersections $\text{adj}(x) \cap \text{Min}(x) = C_x$, $\forall x \in V$, are computed. This computation can be done in constant time $O(1)$ with n processors. Therefore, the substep is executed in time $O(1)$ with n^2 processors. Obviously, the whole substep is executed in time $O(1)$ with n^2 processors.

STEP 4. Finding the set C_x such that $C_y \subseteq C_x$ or $C_y \cap C_x = \emptyset$, for every vertex $y \in AL(v, \ell)$, $1 \leq \ell \leq L_v$, requires time $O(1)$ and $|AL(v, \ell)|^2 \cdot n$ processors. The number of adjacency-level sets that are involved in each execution of this step is obviously less than $n = |V|$. Let k be the number of the adjacency-level sets in the i^{th} execution of the step, and let $n_1, n_2, \ldots, n_k$ be the number of vertices of the 1^{st}, $2^{\text{nd}}, \ldots, k^{\text{th}}$ adjacency-level set, respectively. Then, the number of processors required is $(n_1^2 + n_2^2 + \cdots + n_k^2) \cdot n$. Since $n_1 + n_2 + \cdots + n_k = n$ and $n_i > 0$, $1 \leq i \leq k$, this implies that $n_1^2 + n_2^2 + \cdots + n_k^2 \leq n^2$. Therefore, this substep is executed in time $O(1)$ with no more than n^3 processors.

STEP 5. In this step, it is verified whether the relation $\text{Label}(x) = \text{Label}(y)$, for any vertices x, y, holds. This operation is executed in constant time $O(1)$ with n^2 processors.

STEP 6. In this step, the cardinality of sets, with at most n elements, is computed. Hence, (see also Substep 3.2) with n^2 processors available the computation takes time $O(\log n)$.

Taking into consideration the complexity of each step of the algorithm and the fact that $L \cdot H > n$, we prove the following theorem.

THEOREM 5.1. *Given a graph $G = (V, E)$, algorithm PARPES runs in time $O(\log L \cdot \log H)$ using $L \cdot H \cdot n^2$ processors on a CRCW-PRAM model, where L and H are the length and height of the graph, respectively.*

PROOF. First, we evaluate the total time complexity of the algorithm. For this purpose it is sufficient to calculate the number of repetitions of Steps 2, 3, and 4. Assume that Steps 2, 3, and 4 are executed k times. During the first execution of Step 2, the initial graph $G = (V, E)$ is partitioned into L adjacency-level sets $AL(v, 0)$, $AL(v, 1), \ldots, AL(v, L)$, where L is the length of

graph G. The height of the initial graph is H. We assume that partition (G, v) gives the length of the graph G, i.e., $L_v = L$.

During the second execution of Step 2, the L graphs $G[AL(v, \ell)]$, $1 \leq \ell \leq L$, with $|AL(v, \ell)| \leq H$, are partitioned concurrently. Let L_1 be the greatest length among the L partitions. During the third execution of Step 2, $L \cdot L_1$ graphs are partitioned concurrently, and we denote by L_2 the maximum length of the resulting partitions. During the k^{th} execution of Step 2, $L \cdot L_1 \cdot \ldots \cdot L_{k-1}$ graphs are partitioned concurrently. Let k be the greatest length among the resulting partitions.

We shall omit the extreme case, in which the graphs $G[AL(v, \ell)]$, $1 \leq \ell \leq L$, are complete, i.e., $L_1 = L_2 = \cdots = L_k = 1$. We consider the worst case in which $L_1 = L_2 = \cdots = L_k = 2$. For this case to hold, it is necessary that, in each partition, Levels 1 and 2 contain the same number of vertices. Let this number be $H_1, H_2, \ldots, H_{k-1}$, for the $2^{\text{nd}}, 3^{\text{rd}}, \ldots, k^{\text{th}}$ repetition of Step 2. Then the following relations hold: $L_1 \cdot H_1 = H$, $L_2 \cdot H_2 = H_1, \ldots, L_k \cdot H_k = H_{k-1}$, and therefore,

$$\begin{aligned} H_1 &= \frac{H}{L_1} \\ H_2 &= \frac{H_1}{L_2} \quad = \frac{H}{L_1 \cdot L_2} \\ &\vdots \qquad\quad \vdots \\ H_k &= \frac{H_{k-1}}{L_k} = \frac{H}{L_1 \cdot L_2 \cdot \ldots \cdot L_k}. \end{aligned}$$

In the k^{th} repetition $H_k = 1$. Therefore, $H = L_1 \cdot L_2 \cdot \ldots \cdot L_k$, and $H = 2^k$ or $k = \log H$. Therefore, the time complexity of Steps 2, 3, and 4 is $O(\log L \cdot \log H)$. This is, actually, the complexity of the algorithm.

In order to determine the number of processors required for the execution of the algorithm, it is sufficient to determine the number of processors required for Step 2, specifically for Substep 2.1. The partition of the initial graph $G = (V, E)$ requires $L \cdot H \cdot n^2$ processors, i.e., $P_0 = L \cdot H \cdot n^2$, where L and H are the length and the height of the initial partition, respectively. During the second execution of Step 2, L graphs $G[AL(v, \ell)]$, $1 \leq \ell \leq L$ are partitioned. Let L_ℓ and H_ℓ be the length and height of the ℓ^{th} graph, respectively. Let $G[AL(v, 1)]$ be the graph with $L_1 \cdot H_1 = \max\{L_1 \cdot H_1, L_2 \cdot H_2, \ldots, L_\ell \cdot H_\ell, \ldots, L_L \cdot H_L\}$. The partition of graph $G[AL(v, \ell)]$, $1 \leq \ell \leq L$, requires $L_\ell \cdot H_\ell \cdot H^2 \leq L_1 \cdot H_1 \cdot H^2$ processors. Therefore, the total number P_1 of processors required for the L partitions of Substep 2.1, during the second execution of Step 2, is

$$P_1 \leq L \cdot L_1 \cdot H_1 \cdot H^2 = L \cdot H \cdot H^2.$$

Since $H < n$, where n is the number of vertices of the initial graph, we turn out that $P_1 < L \cdot H \cdot n^2$. Therefore, the second execution of Step 2 requires a smaller number of processors than $L \cdot H \cdot n^2$. Likewise it is shown that the k^{th} repetition of Step 2 requires a number of processors that is less than $L \cdot H \cdot n^2$.

Therefore, the total number of processors required for the execution of the algorithm is $L \cdot H \cdot n^2$, as claimed. ∎

6. ILLUSTRATION OF THE ALGORITHM BY AN EXAMPLE

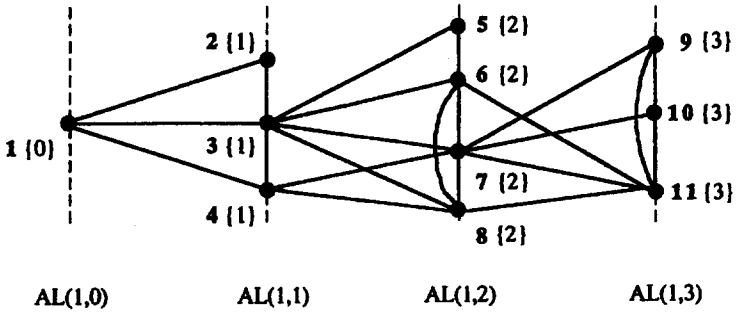
In order to illustrate the workings of algorithm PARPES, we present with the help of an example, the processes of partition, labelling and numbering. The numbers written in brackets beside the vertices are the labels. Here, vertices are named by positive integers.

Input A triangulated graph $G = (V, E)$; Initially, all vertices have an empty label;

1st Execution of Step 2:

$$SV = \{1\} \qquad \text{Equal}(1) = V$$

Partition graph G[Equal(1)], with start vertex $1 \in V$.

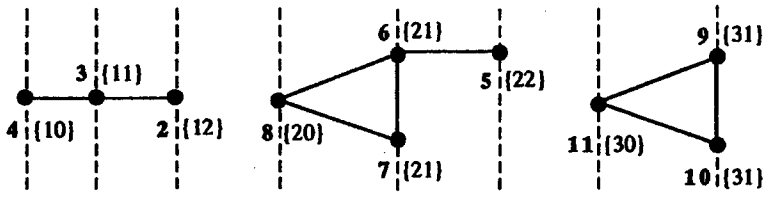


In the next execution of Step 2, the graphs with vertex sets $AL(1,1) = \{2,3,4\}$, $AL(1,2) = \{5,6,7,8\}$ and $AL(1,3) = \{9,10,11\}$ are considered. We select one vertex from each of the vertex sets (see Observation 3, Section 4). We can select Vertex 2 or 3 or 4 from $AL(1,1)$, Vertex 7 or 8 from $AL(1,2)$ and Vertex 11 from $AL(1,3)$. We select Vertices 4, 8 and 11.

2nd Execution of Step 2:

$$\begin{aligned} SV = \{4,8,11\} \qquad & \text{Equal}(4) = \{2,3,4\} \\ & \text{Equal}(8) = \{5,6,7,8\} \\ & \text{Equal}(11) = \{9,10,11\}. \end{aligned}$$

Partition graphs G[Equal(4)], G[Equal(8)], and G[Equal(11)], with start Vertices 4, 8, and 11, respectively.

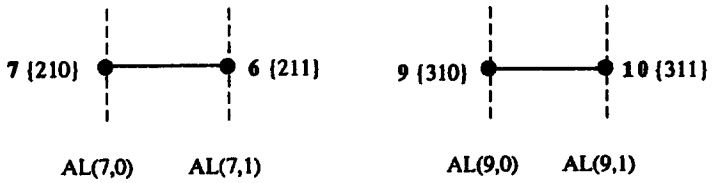


In the next execution of Step 2, the graphs with vertex sets $AL(8,1) = \{6,7\}$ and $AL(11,1) = \{9,10\}$ are considered. We can select Vertex 6 or 7 from $AL(8,1)$ and Vertex 9 or 10 from $AL(11,1)$. We select Vertices 7 and 9.

3rd Execution of Step 2:

$$\begin{aligned} SV = \{7,9\} \qquad & \text{Equal}(7) = \{6,7\} \\ & \text{Equal}(9) = \{9,10\} \end{aligned}$$

Partition graphs G[Equal(7)] and G[Equal(9)], with start Vertices 7 and 9, respectively.



After 3rd execution of Step 2, all labels are deferent, i.e., Label(1) $\neq$ Label(2) $\neq \cdots \neq$ Label(11), and therefore, the control moves to Step 6. The lexicographical order (increasing order) according to their label is a follows:

$$\mathbf{1}\{0\}, \mathbf{4}\{10\}, \mathbf{3}\{11\}, \mathbf{2}\{12\}, \mathbf{8}\{20\}, \mathbf{7}\{210\}, \mathbf{6}\{211\}, \mathbf{5}\{22\}, \mathbf{11}\{30\}, \mathbf{9}\{310\}, \mathbf{10}\{311\}.$$

STEP 6. The ordering σ_R is computed, i.e.,

$$\sigma_R = [\mathbf{1}, \mathbf{4}, \mathbf{3}, \mathbf{2}, \mathbf{8}, \mathbf{7}, \mathbf{6}, \mathbf{5}, \mathbf{11}, \mathbf{9}, \mathbf{10}]$$

and final the reverse ordering of σ_R is computed, i.e.,

$$\sigma = [\mathbf{10}, \mathbf{9}, \mathbf{11}, \mathbf{5}, \mathbf{6}, \mathbf{7}, \mathbf{8}, \mathbf{2}, \mathbf{3}, \mathbf{4}, \mathbf{1}],$$

which is a p.e.s., according to definition mentioned in introduction.

7. CONCLUSIONS

In this paper, a parallel algorithm which computes a Perfect Elimination Scheme of a triangulated graph $G = (V, E)$, is presented. The algorithm, which is based mainly on the properties of the adjacency-level sets, runs in time $O(\log L \cdot \log H)$ using $L \cdot H \cdot n^2$ processors on a CRCW-PRAM model. The partition $\Delta(G, v)$, $v \in V$, of a graph G introduces two characteristic measures in the graph G, which are the length and the height of the graph, L and H, respectively. Note that $L < n$ and $H < n$, where $n = |V|$.

The algorithm can easily be modified to be executed in time $O((\log n + L) \cdot \log H)$ using n^2 processors in the same type of computational model. If we use the partition algorithm ALS_1 of [3] to produce the adjacency-level sets of the graph, then Step 2.1 of the algorithm is executed in time $O(L)$ with n^2 processors. It is of great interest that, if $L \leq \log n$ and the partition algorithm ALS_1 is used, the algorithm is executed in time $O(\log n \cdot \log H)$ with n^2 processors.

REFERENCES

1. J. Naor, M. Naor and A. Schaffer, Fast parallel algorithms for chordal graphs, *Proc.* 19th *Ann. ACM Symp. on Theory of Computing*, pp. 355–364, (1987).
2. C.-W. Ho and R.C.T. Lee, Counting clique trees and computing perfect elimination schemes in parallel, *Inform. Process. Lett.* **31**, 61–68 (1989).
3. S.D. Nikolopoulos and S.D. Danielopoulos, Adjacency-level sets: A constructive tool for designing fast parallel graph algorithms, TR-93-229, Department of Mathematics, University of Ioannina, Greece, (1993).
4. M.C. Golumbic, *Algorithmic Graph Theory and Perfect Graphs*, Academic Press, New York, (1980).
5. A. Edenbrandt, Chordal graph recognition is in NC, *Inform. Process. Lett.* **24**, 239–241 (1987).
6. C.-W. Ho and R.C.T. Lee, Efficient parallel algorithms for maximal cliques, clique tree, and minimum colouring on chordal graphs, *Inform. Process. Lett.* **28**, 301–309 (1988).
7. S. Wagon, Infinite triangulated graphs, *Discrete Math.* **22**, 183–189 (1978).
8. E. Dalhaus and M. Karpinski, Fast parallel computation of perfect and strongly perfect elimination schemes, Res. Rept., RJ 5901 (59206), IBM Research Division, (1987).
9. D.J. Rose, Triangulated graphs and the elimination process, *J. Math. Anal. Appl.* **32**, 597–609 (1970).
10. T. Ohtsuki, A fast algorithm for finding an optimal ordering for vertex elimination on a graph, *SIAM J. Comput.* **5**, 133–145 (1976).
11. D.J. Rose, E. Tarjan and G.S. Lueker, Algorithmic aspects of vertex elimination on graphs, *SIAM J. Comput.* **5**, 266–283 (1976).
12. R.E. Tarjan and M. Yannakakis, Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs, *SIAM J. Comput.* **13**, 566–579 (1984).
13. R.E. Tarjan and M. Yannakakis, Addendum to [12], *SIAM J. Comput.* **14**, 254–255 (1985).
14. S.D. Nikolopoulos, Parallel Recognition and Location Algorithms for Chordal Graphs using Distance Matrices, *Lecture Notes in Computer Science*, Volume 854, pp. 349–358, Springer-Verlag, (1994).
15. P. Beame and J. Hastad, Optimal bounds for decision problems on the CRCW PRAM, *J. Assoc. Comput. Mach.* **36**, 643–670 (1989).
16. F.E. Fich, P.L. Ragde and A. Wigderson, Relations between concurrent-write models of parallel computation, *Proc.* 3rd *Annual ACM Symposium on Principles of Distributed Computing*, pp. 179–189, (1984).
17. L. Kucera, Parallel computation and conflicts in memory access, *Inform. Process. Lett.* **14**, 93–96 (1982).
18. U. Vishkin, Implementation of simultaneous memory address access in model that forbid it, *J. Algorithms* **4**, 45–50 (1983).